Web Service API

646.862.5210
techsupport@puresend.com

www.puresend.com
rev.12.1.2017

v 9.30.0

Getting Started

Appendix

1. Message Deployment

A. createJob

B. abortJob

C. deleteJob

D. getJob

E. startJob

F. updateJob

G. sendTestJob

H. unpauseJob

I. pauseJob

J. updatePureTestJob

K. testFiltering

2. List Management

A. createList

B. importToListViaFTP

C. getListId

D. getListInfo

E. getListInfos

F. deleteList

G. getLists

H. addContactToList

I. getContact

J. getContacts

K. deleteContact

L. setContactOptOutStatus

M. findContactIds

N. getCategoryNames

O. getCategories

P. getRuleSets

Q. createSuppList

Q. createSuppList

R. editSuppList

S. getListEntries

3. Message Creation

A. createMessage

B. createMessageFromURL

C. getMessage

D. updateMessage

E. updateMessageFromURL

4. Job Reporting

A. getJobReport

B. getJobReportDetail

C. getJobIds

puresend

Getting Started

A Quick Start guide to using Purecast API

646.862.5210
techsupport@puresend.com

v 9.30.0

1. Overview

This is the Puresend Web Service API that enables clients to send emails, manage lists and messages via a web service interface. This document assumes you will be using Java client code to connect with the Web Service.

1.1 Security

All of Puresend's servers use secure HTTP (HTTPS), consequently the client will need to locally install the Puresend public SSL certificate for the Web Server they are connecting to. See 2. Setup SSL Certificate. Puresend Support will need to set PURESEND_WEB_SERVICE Permission for an acccount before the Web Service functionality is activated.

1.2 Connecting to Web Service

You can obtain the WSDL at:
<pre>https://<servername>/services/Puresend?wsdl</pre>
The endpoint is defined at the bottom of this WSDL document.

1.3 Sending Emails

In order to send an email you will need to have a List, a Message and a Job. You will then create a job and start it.

2. Setup SSL Certificate

Before you can make calls to the Puresend Web Service you will need to import the Puresend server's SSL certificate into your JVM keystore. To do this, follow the following steps:

- Go to the Puresend Server you will be connecting to.

```
>> https://psip.puresend.com
```

- Select the lock icon at the bottom of the browser. A window will display the certificate information. Find the Export certificate option and save the certificate locally. Save as: `psip.puresend.com.crt`
- Go to your Java bin directory
- Add the certificate to Java keystore (cacerts file) using the java keytool command:

```
>> keytool -import -alias puresend-psip -keystore  <jre path>\lib\se-
curity\cacerts -file <path to saved psip.puresend.com.crt file>
```

```
>> sudo keytool -import -alias puresend-psip -keystore  <jre path>\lib\
security\cacerts -file <path to saved psip.puresend.com.crt file>
```

- If you are asked for a password/passphrase, the default is: *changeit*

3. Creating a List

- To Create a List via the Web Service, use the function:

```
createList()
```

- If you are using custom field names, do not specify ‘Email’ as this is already defined by Puresend and has its on setter function.
- Passing in ‘null’ for the field names will use the default Puresend list master info.
- Sample java code:

```
public  void createListSample() throws java.lang.Exception {
          PuresendStub psStub = new PuresendStub();
          int listId = -1 ;

          // This List is meant to be new
          CreateList tmpCreList = new PuresendStub.CreateList();
          // This puresend account is assumed to exist
          tmpCreList.setUsername(“XXXX”);
          tmpCreList.setPassword(“XXXX”);

          tmpCreList.setListname(“Enter Non Existing Name”);
          tmpCreList.setBrandname(“Enter Brand Name”);

          // Create an array of String to send to the setFieldNames
method
          String[] listFieldValues = new String[]{“FirstName”,”LastNam
e”,”Address”};
          tmpCreList.setFieldNames(listFieldValues);
          tmpCreList.setIsProofList(false);
```

3. Creating a List - contd...

```
          // Invoke the createList call and save the results.
          CreateListResponse tmpCreListRes = psStub.
createList(tmpCreList);
          listId = tmpCreListRes.get_return() ;
          if ( listId > 0) {
               System.out.println("List has been created with id:"
+ listId);
               // Add contacts to created list
               addContactToListSample(listId);
          }
}
```

4. Adding Emails to a List

Adding emails or contacts can be accomplished through the Web Service API.

- To add a Contact/Email via the Web Service, use the function:

```
addContactToList()
```

- Sample java code:

```
public void addContactToListSample(int listId) throws java.lang.
Exception {
     try {
          PuresendStub psStub = new PuresendStub();
          // Add new contact to created list
          AddContactToList tmdAddContToList = new PuresendStub.
AddContactToList();
          // This puresend account is assumed to exist.
          tmdAddContToList.setUsername("XXXX");
          tmdAddContToList.setPassword("XXXX");

          // Use any existing list id
          tmdAddContToList.setListId(listId);
          tmdAddContToList.setEmail("test@test.com");

          // Create an array of String to send to the setFieldNames
method
          // This array length should equal to field of list
          String[] actfieldValues = new String[]
{"value1","value2","value3"};
          tmdAddContToList.setFieldValues(actFieldValues);

          AddContactToListResponse tmdAddContToListRes =
psStub.addContactToList(tmdAddContToList);
```

4. Adding Emails to a List - contd...

```
          if (tmdAddContToListRes.get_return() != null) {
               Contact tmpCont = tmdAddContToListRes.get_return();
               System.out.println("Contact has been added to list with
id:" + tmpCont.getId());
          }

}
```

5. Creating a Message

Creating an Email Message can be accomplished through the Web Service API.

- To create a new Message via the Web Service, use the function:

```
createMessage() OR createMessageFromURL()
```

- Sample java code:

```
public void createMessageSample() {
     try {
          PuresendStub psStub = new PuresendStub();

          // This message is meant to be new
          CreateMessage tmpCreMsg = new CreateMessage();
          // This puresend account is assumed to exist
          tmpCreMsg.setUsername("XXXX");
          tmpCreMsg.setPassword("XXXX");

          tmpCreMsg.setName("Message Name 1");
          // Use -1 for root category Or id of existing category
          tmpCreMsg.setCategoryId(-1);

          // At least one mime type  must contain a non empty string
          tmpCreMsg.setTextContent("Enter Text Content");
          tmpCreMsg.setHtmlContent("Enter HTML Content");
          tmpCreMsg.setMobileContent("Enter Mobile Content");

          // Invoke the createMessage call and save the results.
          CreateMessageResponse tmpCreMsgRes = psStub.
createMessage(tmpCreMsg);
          if (tmpCreMsgRes.get_return() > 0) {
               System.out.println("Message has been created with id:"
                         + tmpCreMsgRes.get_return());
```

5. Creating a Message - contd...

```
          (Exception ex) {
          System.out.println("An error has occurred." +
ex.getMessage());
     }
}
```

6. Creating a Job

Creating a Job can be accomplished through the Web Service API. This creates a Draft Job in the Puresend system. You can then update this draft job with the necessary information.

- To create a Job via the Web Service, use the function:

```
createJob()
```

- To update the necessary information for the Job via the Web Service, use the function:

```
updateJob()
```

- Sample java code:

```
public void createJobSample() {
     try {
          PuresendStub psStub = new PuresendStub();
          int jobId = -1;

          CreateJob tmpCreJob = new CreateJob();
          // This puresend account is assumed to be exist
          tmpCreJob.setUsername("XXXX");
          tmpCreJob.setPassword("XXXX");

          tmpCreJob.setJobName("Enter Non Existing Name");
          // Use -1 for root category Or id of existing category
          tmpCreJob.setCategoryId(-1);
```

6. Creating a Job - contd...

```
// Invoke the createJob call and save the results.
          CreateJobResponse tmpCreJobRes = psStub.createJob(tmpCreJob);
          job_id = tmpCreJobRes.get_return();
          if (jobId > 0) {
               System.out.println("Job has been created with id:"
+ jobId);

               UpdateJob tmpUpJob = new UpdateJob();
               // This puresend account should be same as above used.
               tmpUpJob.setUsername("XXXX");
               tmpUpJob.setPassword("XXXX");

               Job tmpJob = new Job();

               // Create an array of String to send to the
setClusterNames method
               String[] tmpClusterNames = new String[] { "clusterA"};
               tmpJob.setClusterNames(tmpClusterNames);
               // Use valid mail address
               tmpJob.setFrom("test@test.com");
               tmpJob.setFriendlyFrom("Enter Value For From");
               // Use new created job id
               tmpJob.setId(jobId);
               // Create an array of int to send to the setListIds
method
               // Use associate broadcast list id's of user
               int[] bListIds = new int[] { 123, 234, 345 };
               tmpJob.setListIds(bListIds);
               // Use existing message id
               tmpJob.setMessageId(123);
               tmpJob.setName("Enter Non Existing Name");
               // Use ruleset id
               tmpJob.setRulesetId(123);
               tmpJob.setName("Enter Non Existing Name");
```

6. Creating a Job - contd...

```
                // Use ruleset id
                tmpJob.setRulesetId(123);
                // Create an instance of Calender to send to the
setScheduledDate method
                // Ignore or Set to null to schedule job immediately
                Calendar tmpCal = Calendar.getInstance();
                tmpJob.setScheduledDate(tmpCal);

                tmpJob.setSubject("Enter Subject");
                // Create an array of int to send to the
setSuppressionIds method
                // Use associate suppress list id's of user
                int[] sListIds = new int[] { 321, 432, 543 };
                tmpJob.setSuppressionIds(sListIds);
                tmpJob.setValues("Enter the value");

                // Associate this job instance to UpdateJob instance
                tmpUpJob.setJobInfo(tmpJob);
                // Invoke the updateJob call and save the results.
                UpdateJobResponse tmpUpJobRes = psStub.
updateJob(tmpUpJob);
                if (tmpUpJobRes.get_return() == 1) {
                     System.out.println("Job has been updated "
                               + tmpUpJobRes.get_return());
                }
           }
     } catch (Exception e) {
          System.out.println("An unexpected error has occurred." +
e.getMessage());
     }
}
```

7. Sending a Test Job

A Test Job can be sent via the Web Service API. You can specify specific email addresses or an existing proof list. If emails and a proof list is specified, the proof list is ignored and the test job will only be sent to the emails. Set emails to NULL if you wish to use a proof list.

- To Send a Test Job via the Web Service, use the function:

```
sendTestJob()
```

- Sample java code:

```
public void sendTestJobSample(int jobId) {
     try {
          PuresendStub psStub = new PuresendStub();

          SendTestJob tmpTestJob = new SendTestJob();
          // This puresend account is assumed to be exist
          tmpTestJob.setUsername("XXXX");
          tmpTestJob.setPassword("XXXX");

          // Use existing "Draft" job id
          tmpTestJob.setJobId(jobId);

          //Option 1: Use any valid email address
          tmpTestJob.setToEmails("test@test.com");
          //Option 2: Use an existing proof list. setToEmails() should
be set to null.
          //Create an array of int of existing proof list ids
          int[] tmpProofLstIds = new int[]{2345};
          tmpTestJob.setProofListIds(tmpProofLstIds);
```

7. Sending a Test Job - contd...

```
          // Invoke the createJob call and save the results.
          SendTestJobResponse tmpTestJobRes = psStub.
sendTestJob(tmpTestJob);
          if (tmpTestJobRes.get_return() == 1) {
               System.out.println("Job has been launch successfuly:" +
tmpTestJobRes.get_return());
          }
     } catch (Exception e) {
          System.out.println("An error has occurred." +
e.getMessage());
     }
}
```

8. Starting a Job

A Job can be started via the Web Service API. The system will verify the job data is correct. If any errors are encountered an error will be thrown with an appropriate error message. Use `updateJob()` to change the job data.

- To start a Job via the Web Service, use the function:

```
startJob()
```

- Sample java code:

```
public void startJobSample() {
     try {
          PuresendStub psStub = new PuresendStub();
          StartJob tmpStartJob = new StartJob();

          // This puresend account is assumed to exist
          tmpStartJob.setUsername("XXX");
          tmpStartJob.setPassword("XXX");

          // Use existing "Draft" job id
          tmpStartJob.setJobId(123);

          StartJobResponse tmpStartJobRes = psStub.
startJob(tmpStartJob);
          if(tmpStartJobRes.get_return() == 1){
               System.out.println("Job has been start successfully.");
          }
     } catch (Exception e) {
          System.out.println("An error has occurred." +
e.getMessage());
     }
}
```

9. Viewing the Job Report

A high level job report can be viewed for the delivery status.

- To view the standard job report via the Web Service, use the function:

```
getJobReport()
```

- Sample java code:

```
public static void getJobReportSample(){
     try {
          PuresendStub psStub = new PuresendStub();
          // Job is meant to be existing for Report
          GetJobReport tmpJobRep = new GetJobReport();

          // This puresend account is assumed to exist
          tmpJobRep.setUsername("XXX");
          tmpJobRep.setPassword("XXX");

          // Use existing job id
          tmpJobRep.setJobId(123);
          GetJobReportResponse tmpJobRepRes = psStub.
getJobReport(tmpJobRep);
          String resXmlString = tmpJobRepRes.get_return();
           if(resXmlString != null){
                System.out.println(resXmlString);
           }
     } catch (Exception e) {
          System.out.println("An error has occurred." +
e.getMessage());
     }
}
```

Appendix

A complete list of Web Service API Calls

1. Message Deployment

A. createJob

```
public int createJob(java.lang.String username,
                     java.lang.String password,
                     java.lang.String jobName,
                     int categoryId)
              throws java.lang.Exception
```

Create a draft job. Use `updateJob()` to set additional job details prior to calling `startJob()`. Note: Job does not begin until calling `startJob()`.

Parameters:

`username` - account login name
`password` - account password
`jobName` - job name. Must be a unique name.
`categoryId` - category to create job in. -1 for default top level category.

Returns:

job id of the newly created job

Throws:

`java.lang.Exception`

B. abortJob

```
public int abortJob(java.lang.String username,
                    java.lang.String password,
                    int jobId)
             throws java.lang.Exception
```

Ability to abort the job when it is paused, NOT finished and the state equals `Ready`.

Parameters:

`username` - account login name
`password` - account password
`jobId` - Job Id

Returns:

success = 1, failed = 0

Throws:

`java.lang.Exception`

C. deleteJob

```
public int deleteJob(java.lang.String username,
                     java.lang.String password,
                     int jobId)
              throws java.lang.Exception
```

Delete a job.

Parameters:

`username` - account login name
`password` - account password
`jobId` - Job Id

Returns:

success = 1, failed = 0

Throws:

`java.lang.Exception`

D. getJob

```
public Job getJob(java.lang.String username,
                  java.lang.String password,
                  int jobId)
           throws java.lang.Exception
```

Obtain the Job information.

Parameters:

`username` - account login name
`password` - account password
`jobId` - Job Id

Returns:

Job information object. Returns null if job does not exist.

Throws:

`java.lang.Exception`

E. startJob

```
public int startJob(java.lang.String username,
                    java.lang.String password,
                    int jobId)
             throws java.lang.Exception
```

Start a job from a draft job based on user privileges.

Parameters:

`username` - account login name
`password` - account password
`jobId` - Job Id

Returns:

success = 1; failed = 0

Throws:

`java.lang.Exception`

F. updateJob

```
public int updateJob(java.lang.String username,
                     java.lang.String password,
                     Job jobInfo)
              throws java.lang.Exception
```

Update a newly created job. Can update the job as long as it is in the draft status. Cannot update the job once it has started. Job data that can be updated using the Job object is as follows: list ids, suppression ids, cluster names, rule set, message id, friendly from, from, subject and scheduled date, Friendly Reply-To, Reply-To, From format, search criteria ID, Skip the first, Use At Most

Parameters:

`username` - account login name
`password` - account password
`job` - the Job object that contains the updated data

Returns:

success = 1; failed = 0

Throws:

`java.lang.Exception`

G. sendTestJob

```
public int sendTestJob(java.lang.String username,
                       java.lang.String password,
                       int jobId,
                       java.lang.String toEmails,
                       int[] proofListIds)
                throws java.lang.Exception
```

Send a test job to the specified email list or proof list. If both email and proof list is specified, test job will be sent to the emails.

Parameters:

`username` - account login name

`password` - account password

`jobId` - Job Id

`toEmails` - comma separated list of emails to send test job to.

`proofListId` - id of proof list to send test job to. Ensure toEmails is set to null.

Returns:

success = 1, fail = 0

Throws:

`java.lang.Exception`

H. unpauseJob

```
public int unpauseJob(java.lang.String username,
                      java.lang.String password,
                      int jobId)
               throws java.lang.Exception
```

Start paused job based on user privileges.

Parameters:

`username` - account login name
`password` - account password
`jobId` - Job Id

Returns:

success = 1; failed = 0

Throws:

`java.lang.Exception`

I. pauseJob

```
public int pauseJob(java.lang.String username,
                    java.lang.String password,
                    int jobId)
             throws java.lang.Exception
```

Pause a job if the job has not failed previously, is not a draft job and has not finished.

Parameters:

`username` - account login name
`password` - account password
`jobId` - Job Id

Returns:

success = 1; failed = 0

Throws:

`java.lang.Exception`

J. updatePureTestJob

```
public int createList(java.lang.String username,
                      java.lang.String password,
                      java.lang.String puretestJobInfo)
              returns success = 1; failed = 0
               throws java.lang.Exception
```

Update a newly created job. Can update the job as long as it is in the draft status. Cannot update job once it has started.

Parameters:

`username` - account login name

`password` - account password

`listname` - name of the list

`brand` - any human readable name used for list purposes category.

`isProofList` - is this list a proof list (true/false)

`fieldNames` - custom field names for the list. Maximum 25 fields allowed.

Returns:

Success = 1; Failed = 0

Throws:

`java.lang.Exception`

K. testFiltering

```
public int createList(java.lang.String username,
                      java.lang.String password,
                      java.lang.String jobId)
               returns result
                throws java.lang.Exception
```

Perform an email filtering test

Parameters:

`username` - account login name
`password` - account password
`jobId` - Job ID

Returns:

Filtering test result in XML format

Throws:

`java.lang.Exception`

2. List Management

A. createList

```
public int createList(java.lang.String username,
                      java.lang.String password,
                      java.lang.String listname,
                      java.lang.String brandname,
                      boolean isProofList,
                      java.lang.String[] fieldNames)
```

Creates a list with name, brand and custom field information.

Parameters:

`username` - account login name

`password` - account password

`listname` - name of the list

`brand` - any human readable name used for list purposes category.

`isProofList` - is this list a proof list (true/false)

`fieldNames` - custom field names for the list. Maximum 25 fields allowed.

Returns:

new list id

Throws:

`java.lang.Exception`

B. importToListViaFTP

```
public int importToListViaFTP(java.lang.String username,
                              java.lang.String password,
                              java.lang.String csvFileName,
                              byte[] csvFileData,
                              byte[] xmlControlFileData)
                       throws java.lang.Exception
```

Imports the CSV list data with the auto import XML file to the user's FTP directory. Ensure the account's FTP directory has been setup.

Parameters:

`username` - account login name

`password` - account password

`csvFileName` - filename used for import. Filename used for .xml and .go file

`csvFileData` - CSV file content data (byte[] array)

`xmlControlFileData` - XML control data. see *https://int.puresend.com/pc/index.php/FTP_Interface#Automated_Upload_Facility*

`fieldNames` - custom field names for the list. Maximum 25 fields allowed.

Returns:

success = 1; failed = 0

Throws:

`java.lang.Exception`

C. getListId

```
public int getListId(java.lang.String username,
                     java.lang.String password,
                     java.lang.String listname)
              throws java.lang.Exception
```

Retrieves the list id for the given list name. If multiple lists exist with the same name, the first list's id is returned.

Parameters:

`username` - account login name
`password` - account password
`listname` - list name to be searched

Returns:

list id corresponding to the list name

Throws:

`java.lang.Exception`

D. getListInfo

```
public ListInfo getListInfo(java.lang.String username,
                            java.lang.String password,
                            int listId)
                     throws java.lang.Exception
```

Get a listInfo object with list id.

Parameters:

`username` - account login name
`password` - account password
`listId` - list id for which data will be returned

Returns:

ListInfo object containing the list information

Throws:

`java.lang.Exception`

E. getListInfos

```
public ListInfo[] getListInfos(java.lang.String username,
                               java.lang.String password,
                               int listType)
                        throws java.lang.Exception
```

Retrieves all the lists including the list information for the user's account.

Parameters:

`username` - account login name
`password` - account password
`listType` – [OPTIONAL] list type. Value:
1 for regular lists
2 for database suppression sists
3 for file-based suppression list
4 for file-based domain suppression list
5 for file-based username suppression list
7 for file-based MD5 suppression list
8 for seed list

Returns:

listInfos array of objects containing the list information

Throws:

`java.lang.Exception`

F. deleteList

```
public int deleteList(java.lang.String username,
                      java.lang.String password,
                      int listId)
               throws java.lang.Exception
```

Deletes a list.

Parameters:

`username` - account login name
`password` - account password
`listId` - list id to be deleted

Returns:

success = 1, failed = 0

Throws:

`java.lang.Exception`

G. getLists

```
public int[] getLists(java.lang.String username,
                      java.lang.String password,
                      int listType)
               throws java.lang.Exception
```

Retrieves all the list ids for the user's account.

Parameters:

`username` – account login name
`password` – account password
`listType` – [OPTIONAL] list type. Value:
1 for regular lists
2 for database suppression sists
3 for file-based suppression list
4 for file-based domain suppression list
5 for file-based username suppression list
7 for file-based MD5 suppression list
8 for seed list

Returns:

array of list ids

Throws:

`java.lang.Exception`

H. addContactToList

```
public Contact addContactToList(java.lang.String username,
                                java.lang.String password,
                                int listId,
                                java.lang.String email,
                                java.lang.String[] fieldValues,
                                boolean allowDuplicates,
                                int actionCode)
```

Add an email/contact with associated fields to the specified list.

Parameters:

`username` - account login name

`password` - account password

`listId` - id of the list to which the email will be added to

`email` - email address to be added

`fieldValues` [OPTIONAL]- array of the field values as specified for the list. This will constitute one entry in the list.

`allowDuplcates` [OPTIONAL] - insert an email even if it exists in the list specified.

`actionCode` [OPTIONAL] - level of how to handle duplicated email.

0 for failure. Default behavior.

1 for return existing record if email existed in the list specified

Throws:

`java.lang.Exception`

I. getContact

```
public Contact getContact(java.lang.String username,
                          java.lang.String password,
                          int contactId)
```

Retrieves email/contact details for the specified contact id.

Parameters:

`username` - account login name
`password` - account password
`contactId` - contact id details that to be retrieved.

Returns:

Contact object containing details of the email and the field values.

Throws:

`java.lang.Exception`

J. getContacts

```
public Contact[] getContacts(java.lang.String username,
                             java.lang.String password,
                             int[] contactIds)
                      throws java.lang.Exception
```

Retrieves email/contact details for the specified contact ids.

Parameters:

`username` - account login name
`password` - account password
`contactIds` - list of contact ids to be retrieved

Returns:

an array of Contact objects

Throws:

`java.lang.Exception`

K. deleteContact

```
public int deleteContact(java.lang.String username,
                         java.lang.String password,
                         int listId,
                         java.lang.String email)
```

Remove an email/contact related to the specified list.

Parameters:

`username` - account login name
`password` - account password
`listId` - id of the list to which the email will be removed from
`email` - email address to be removed

Returns:

success = 1, failed = 0

Throws:

`java.lang.Exception`

L. setContactOptOutStatus

```
public int setContactOptOutStatus(java.lang.String username,
                                  java.lang.String password,
                                  int contactId,
                                  boolean hasOptedOut)
```

Set opt-out status for the specified email contact

Parameters:

`username` - account login name
`password` - account password
`contactId` - email/contact to set the opt-out status for
`hasOptOut` - whether contact has opted out (true = opted out)

Returns:

success = 1, failed = 0

Throws:

`java.lang.Exception`

M. findContactIds

```
public int[] findContactIds(java.lang.String username,
                            java.lang.String password,
                            java.lang.String email,
                            boolean searchSubAccounts)
```

Retrieves a list of contact IDs by email address. Option to include searching sub-accounts.

Parameters:

`username` - account login name
`password` - account password
`email` - email search string
`searchSubAccounts` - whether to search sub accounts

Returns:

contact IDs array

Throws:

`java.lang.Exception`

N. getCategoryNames

```
public java.lang.String[] getCategoryNames(java.lang.String username,
                                           java.lang.String password,
                                           int[] categoryIds)
```

Retrieves a list category names for the specified category ids. Invalid category ids will return a category name with an invalid name descriptor.

Parameters:

`username` - account login name
`password` - account password
`categoryIds` - array of category ids

Returns:

array of category names

Throws:

`java.lang.Exception`

O. getCategories

```
public CategoryInfo[] getCategories(java.lang.String username,
                                    java.lang.String password,
                                    int categoryType)
```

Retrieves a list of categories by category type

Parameters:

`username` - account login name

`password` - account password

`categoryType` - category Type. Value:

3 for regular messages
4 for template messages
5 for confirmation messages
6 for landing page messages
7 for jobs
8 for lists
9 for seed lists

Returns:

an array of CategoryInfo Objects

Throws:

`java.lang.Exception`

P. getRuleSets

```
public RuleSetInfo[] getRuleSets(java.lang.String username,
                                 java.lang.String password)
```

Retrieves a list of rulesets.

Parameters:

`username` - account login name

`password` - account password

Returns:

an array of RuleSetInfo Objects

Throws:

`java.lang.Exception`

Q. createSuppList

```
public int createSuppList (java.lang.String username,
                           java.lang.String password,
                           java.lang.String listname,
                           java.lang.String brandName,
                           int type
                           java.lang.String filename)
```

Creates a suppression list with name, brand and file name information

Parameters:

`username` - account login name
`password` - account password
`listname` - name of the list
`brandName` - any human readable name used for list purposes
`type` - suppression list type
`fileName` - if this was a file-based suppression list, filename should be presented

Returns:

new list id

Throws:

`java.lang.Exception`

R. editSuppList

```
public int editSuppList(java.lang.String username,
                        java.lang.String password,
                        int listId,
                        java.lang.String listName,
                        java.lang.String brandName,
                        java.lang.String fileName)
```

Edit a suppression list with name, brand and filename information

Parameters:

`username` - account login name
`password` - account password
`listId` - ID of the suppression list
`listName` - name of the suppression list
`brandName` - any human readable name used for list purposes
`fileName` - if this was a file-based suppression list, filename should be presented

Returns:

existing list id

Throws:

`java.lang.Exception`

S. getListEntries

```
public int getListEntries(java.lang.String username,
                          java.lang.String password,
                          int listId,
                          java.lang.String offset,
                          java.lang.String limit)
```

Retrieves email details for the specified List

Parameters:

`username` - account login name
`password` - account password
`listId` - ID of the list
`offset` - parameter can be passed to offset in the results
`limit` - parameters can be passed to limit the number of returned contacts

Returns:

string object containing one list recipient per line.
Format of each line:
(Entry Id, List Id, Email, Import Date, Last Mail Date, Bounced, opt-out, Temp bonces, Email Format, Custom Fields size, Custom Fields(if any))

Throws:

`java.lang.Exception`

3. Message Creation

A. createMessage

```
public int createMessage(java.lang.String username,
                         java.lang.String password,
                         java.lang.String name,
                         int categoryId,
                         java.lang.String htmlContent,
                         java.lang.String textContent,
                         java.lang.String mobileContent,
                         int headerId,
                         int footerId)
```

Creates a Content Message for use in a job.

Parameters:

`username` - account login name
`password` - account password
`name` - name of content message
`categoryId` - category to create Message in. -1 for default top level category.
`htmlContent` - message content for html based viewing
`textContent` - message content for text viewing allowed.
`mobileContent` - message content for mobile viewing allowed.
`headerId` – content ID of header
`footerId` – content ID of footer

Returns:

the newly created message id

Throws:

`java.lang.Exception`

B. createMessageFromURL

```
public int createMessageFromURL(java.lang.String username,
                                 java.lang.String password,
                                 java.lang.String name,
                                 int categoryId,
                                 java.lang.String htmlContentURL,
                                 java.lang.String textContentURL,
                                 java.lang.String mobileContentURL
                                 int headerId,
                                 int footerId)
```

Sets and post-processes the specified content type to the contents of the specified URL and creates a Message. Fails if the MIME type returned by the URL doesn't match the requested MIME type.

Parameters:

`username` - account login name
`password` - account password
`name` - name of content message
`categoryId` - category to create Message in. -1 for default top level category.
`htmlContentURL` - message content URL for html based viewing
`textContentURL` - message content URL for text viewing allowed.
`mobileContentURL` - message content URL for mobile viewing allowed.
`headerId` – content ID of header
`footerId` – content ID of footer

Returns:

the newly created message id

Throws:

`java.lang.Exception`

C. getMessage

```
public Message getMessage(java.lang.String username,
                          java.lang.String password,
                          int messageId)
```

Retrieves Message details for specified message id and includes rendered data.

Parameters:

`username` - account login name
`password` - account password
`messageId` - the message id to be retrieved

Returns:

Message detailed information

Throws:

`java.lang.Exception`

D. updateMessage

```
public int updateMessage(java.lang.String username,
                         java.lang.String password,
                         Message message,
                         java.lang.String mimeType
                         int headerId,
                         int footerId)
                  throws java.lang.Exception
```

Updates an existing Message based on the Mime Type.

Parameters:

`username` - account login name
`password` - account password
`message` - new message to replace existing message
`mimeType` - for the message version to be updated. Valid types are: “text/html”, “text/plain”, “text/x-html-mobile”.
`headerId` – content ID of header
`footerId` – content ID of footer

Returns:

success = 1 otherwise, -1.

Throws:

`java.lang.Exception`

E. updateMessageFromURL

```
public int updateMessageFromURL(java.lang.String username,
                                java.lang.String password,
                                int messageId,
                                java.lang.String htmlContentURL,
                                java.lang.String textContentURL,
                                java.lang.String mobileContentURL
                                int headerId,
                                int footerId)
                          throws java.lang.Exception
```

Sets and post-processes the specified content type to the contents of the specified URL and update an existing Message Object. Fails if the MIME type returned by the URL doesn't match the requested MIME type.

Parameters:

`username` - account login name
`password` - account password
`message` - new message to replace existing message
`htmlContentURL` - message content URL for html based viewing
`textContentURL` - message content URL for text based viewing
`mobileContentURL` - message content URL for mobile based viewing
`headerId` – content ID of header
`footerId` – content ID of footer

Returns:

success = 1 otherwise, -1.

Throws:

`java.lang.Exception`

4. Job Reporting

A. getJobReport

```
public java.lang.String getJobReport(java.lang.String username,
                                      java.lang.String password,
                                      java.lang.String name,
                                      int jobId)
```

Returns standard job report represented as a XML formatted string.

Parameters:

`username` - account login name
`password` - account password
`name` - name of content message
`jobId` - job id

Returns:

job report in XML format

Throws:

`java.lang.Exception`

B. getJobReportDetail

```
public java.lang.String getJobReportDetail(java.lang.String username,
                                            java.lang.String password,
                                            int jobId, int type)
```

Returns standard job report detail represented as a XML formatted string.

Parameters:

`username` - account login name
`password` - account password
`jobId` - job id
`type` - report type. Value:

2: Clickthru
3: Total opens
4: Opt out
8: Permanent bounce
9: Temporary bounce
10: Blocked bounce
36: Feedback loop complaints
38: Queued
40: Delivered

Returns:

job report in XML format

Throws:

`java.lang.Exception`

C. getJobIds

```
public java.lang.String getJobIds(java.lang.String username,
                                   java.lang.String password,
                                   java.lang.String startDate,
                                   java.lang.String endDate)
```

Returns standard job report detail represented as a XML formatted string.

Parameters:

`username` - account login name
`password` - account password
`start date` - the start date of the date range
`end date` - the end date of the date range

Returns:

Job ID array. Returns null if job does not exist in XML format

Throws:

`java.lang.Exception`

Web Service API

For additional information,
please visit www.puresend.com

or call our support line at 646.862.5210

or email our support staff at
techsupport@puresend.com